Contents

Chapter 1

Conclusion and Future Work

WebAssembly has become a new tehcnology for web browsers and standalone engines such as the ones used in Edge-Cloud platforms. WebAssembly is designed with security and sandboxing premises, yet, is still vulnerable. Besides, since it is a relatively new technology, new vulnerabilities appear in the wild every, a t a higher pace than the adoption of patches and defenses. Software diversification, as a widely studied field, could be a solution for known and yet-unknown vulnerabilities. However, none research on this field has been conducted for WebAssembly.

In this work, we propose an automatic approach to generate software diversification for WebAssembly. Our approaches are based on automated software transformations highlighted by an exhaustive literature research. In addition, we provide the complementary implementation for our approaches, including a generic LLVM superdiversifier that could be used to extend our ideas to other programming languages. We empirically demonstrate the impact of our approach by providing Randomization and Multivariant Execution (MVE) for WebAssembly. For this, we provide two tools, CROW and MEWE. CROW completely automatizes the process by using a superdiversifier. MEWE provides execution path randomization for an MVE. This chapter is organized into two sections. In Section 1.1, we summarizes the main results we found by answering our research questions enunciated in **??**. Finally, Section 1.2 describes potential future work that could extend this dissertation.

1.1 Summary of the results

We enunciate the three research questions in **??**. With the first research question we investigate the static properties of the software diversification for WebAssembly generated by our approaches. We answer our first research question by creating programs variants for 3021 original programs. With CROW, we create program variants for the 11.78% of the programs in our corpora. We study the properties of the generated variants at the level of generated programs' population. Thus,

we identify the challenges that attempt against the generation of unique program variants. Besides, we highlight the code properties that offers high number of program variants.

Complementary with our first research question, we evaluate the dynamic properties of the program variants generated in the answering of our first research question. We execute each of the 303 original program and its generated variants for the Rosetta. For each execution, we collect their execution trace and its execution times. We demonstrate that the WebAssembly variants generated by CROW offer remarkable different execution traces. Similarly, the execution times are different between each program and its variants. For the 71% of the diversified programs, at least one variant has an execution time distribution that is different compared to the execution time distribution of the original program. Moreover, CROW generates both faster and slower variants. Nevertheless, we highlighted the importance of dynamic analysis for software diversification.

Our last and third research question evaluates the impact of providing a world-wide MVE for WebAssembly. We use MEWE to build multivariant binaries for the program variants generated for Libsodium and QrCode corpora. We deploy the generated multivariant binaries in an Edge-Cloud platform, collecting their execution times. The addition of runtime path randomization to multivariant binaries provides significant differences between the execution of the original binary and the multivariant binary. The observed differences lead us to conclude that no specific variant can be inferred from studying the execution time of the multivariant binaries. Therefore, attacks relying on measuring precise execution times are made harder to conduct.

These results, overall, show that our approaches are able to provide an automated end-to-end solution for the diversification of WebAssembly programs. Our approaches provide different observable properties that are commonly used to conduct attacks such as: static code analysis, execution traces and execution time. Therefore, our approaches can be used to harden unknown and yet-unknown vulnerabilities.

1.2 Future work

Along with this dissertation we highlighted challenges and limitations. In all cases we proposed solutions, yet, some of them could be explored more in depth as a call for optimization. For example, as we mentioned in **??** our solution provides program variants, but remarkably lower unique variants as a consequence of the replacement combining process of CROW (**??**). Techniques relying on smart heuristics [?] could help to increase the generation of unique variants by early discarding unsound combinations. On the other hand, constant inferring does not always finish in a successful replacement, due to the CROW's obliviousness for some computation models, such as memory operations. A solution could be also to use heuristics to select which part of the code is more probable to become a constant assignment.

As we mentioned in **??**, another approach to provide software diversification for Wasm could be binary-to-binary (Wasm to Wasm) transformations [?] . This approach could be used to increase resilience in malware classifiers through the study of diversification as an obfuscation technique [**?**]. Obfuscation of Wasm code could be used to measure the accuracy of malware classifiers.

By using a superdiversifier, the generated code transformations outperform hand-written transformations. An evidence of this is the CVE[1] found in the code generation component of wasmtime. We found this CVE during the implementation of MEWE with one of the generated variants. This highlighted the need for better strategies of stressing compilers, interpreters and validators of Wasm. CROW and MEWE can be included in fuzzing campaigns [?] to provide better testing and preventing vulnerabilities.

[1] `https://www.fastly.com/blog/defense-in-depth-stopping-a-wasm-compiler-bug-before-it-became-a-problem`

Bibliography

[] Bhansali,S., Aris,A., Acar,A., Oz,H., and Uluagac,A. S. (2022). A first look at code obfuscation for webassembly. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '22, page 140–145, New York, NY, USA. Association for Computing Machinery.